# A Comparison between Two Massively Parallel Algorithms for Monte Carlo Computer Simulation: An Investigation in the Grand Canonical Ensemble

**GRANT S. HEFFELFINGER***

*Sandia National Laboratories, P.O. Box 5800, Albuquerque, New Mexico 87185-1111.*
*E-mail: gsheffe@sandia.gov*

**MARTIN E. LEWITT**

*P.O. Box 729, Sandia Park, New Mexico 87047-0729.*
*E-mail: lewitt@swcp.com*

*Received 4 October 1994; accepted 10 May 1995*

## ABSTRACT

We present a comparison between two different approaches to parallelizing the grand canonical Monte Carlo simulation technique (GCMC) for classical fluids: a spatial decomposition and a time decomposition. The spatial decomposition relies on the fact that for short-ranged fluids, such as the cut and shifted Lennard-Jones potential used in this work, atoms separated by a greater distance than the reach of the potential act independently, and thus different processors can work concurrently in regions of the same system which are sufficiently far apart. The time decomposition is an exactly parallel approach which employs simultaneous (GCMC) simulations, one per processor, identical in every respect except the initial random number seed, with the thermodynamic output variables averaged across all processors. While scaling characteristics for the spatial decomposition are presented for 8–1024 processor systems, the comparison between the two decompositions is limited to the 8–128 processor range due to the warm-up time and memory imitations of the time decomposition. Using a combination of speed and statistical efficiency, the two algorithms are compared at two different state points. While the time decomposition reaches a given value of standard error in the system's potential energy more quickly than the spatial decomposition for both densities, the warm-up time demands of the time decomposition quickly become insurmountable as the system size increases.


*Author to whom all correspondence should be addressed.



CCC 0192-8651 / 96 / 020250-16

## Introduction

Molecular dynamics (MD) has proven relatively easy to adapt to parallel architectures largely due to the fact that the bulk of the computational work of an MD simulation—the calculation of the forces and new positions of the atoms—is performed for each atom in the system at the same time. Parallel Monte Carlo (MC) simulations, on the other hand, have been more difficult to achieve because atoms in an MC simulation are moved one at a time. While the bulk of the computational work in an MC simulation (the energy calculation of a proposed move) can be shared across processors, given that many MC simulations employ short-range potentials, this approach is inefficient because only a small subset of the system's atoms interact with any one atom at any given point in the simulation. Therefore, any efficient massively parallel Monte Carlo algorithm for systems with short-range interactions must involve simultaneous moves on different processors.

This work focuses on a comparison between two parallel Monte Carlo algorithms suitable for short-range interactions which have been applied to Monte Carlo simulations in the grand canonical ensemble (constant volume $V$, chemical potential $\mu$, and temperature $T$). However, both algorithms are equally well suited for application to $NVT$ Monte Carlo (constant number of atoms $N$, volume $V$, and temperature $T$).

## Parallel Monte Carlo Algorithms

Monte Carlo simulations have been parallelized in several different ways. The approaches fall into two different categories depending on whether or not the parallel algorithm relies on modifying the simulation's Markov chain to make it parallelizable.

### PARALLEL MONTE CARLO ALGORITHMS WITH MODIFIED MARKOV CHAINS

As stated earlier, any efficient massively parallel Monte Carlo algorithm for short-range interactions must involve simultaneous moves on different processors. This can be accomplished by modifying the Monte Carlo algorithm itself to yield a new, parallelizable algorithm. An example of this approach is parallel multiparticle Monte Carlo, which simply involves proposing new positions for some subset $K$ of the system's $N$ atoms, calculating the energy of this composite move in parallel with a maximum of $K$ processors, and accepting/rejecting the composite move. The major difficulty with this method is that the computational speedup enabled by the parallelizable modified Markov chain is offset by its reduced statistical efficiency.[1] (This raises an important point that we will return to later: A true measure of the speed of a Monte Carlo simulation includes not only a measure of its algorithm's computational speed but also its statistical efficiency.) A second approach to parallelizing MC which employs a modified Markov chain relies on the fact that in simulations of short-range potentials, atoms separated by a distance greater than the range of the potential are independent and can therefore be moved simultaneously. Johnson[2] took advantage of this fact to parallelize an $NVT$ MC simulation for a two-dimensional Lennard-Jones system with a cutoff of $3.0\sigma$. Each processor, assigned a region of the simulation volume, moved an atom in its volume at the same time, calculating the energy of its proposed move as well as that of the proposed moves of its neighbors. When adjacent processors moved atoms within range of each other, the move initiated with the earlier clock time was assigned priority and completed first, followed by the second move. The main disadvantage of this method is that the inefficiencies due to this contention between neighboring processors will increase in three dimensions and at higher densities.

### PARALLEL MONTE CARLO ALGORITHMS WITH UNMODIFIED MARKOV CHAINS

The most obvious approach to parallelizing Monte Carlo simulations, what we call time decomposition, is the best example of parallel Monte Carlo algorithms which do not rely on modified Markov chains. In this method, the computational power of the parallel architecture is used to run simultaneous simulations, one per processor, identical in every respect except the initial value of the random number generator seed.[3] The desired results of the simulation are then averaged over all processors, yielding better statistics than a single simulation of the same duration. This approach may be thought of as a decomposition over Monte Carlo time because stepping through a sequence of random numbers in a Monte Carlo simulation is

analogous to stepping through time in a molecular dynamics simulation, with each processor in the Monte Carlo time decomposition starting at a different point in time. The time decomposition is easiest to program but has two idiosyncrasies which, depending on the system to be studied, may make it inappropriate or even impossible to use. The first problem occurs for Monte Carlo simulations which require a significant warm-up phase before equilibrium data can be accumulated. If this phase is not short relative to the production phase (in which the desired equilibrium data are accumulated), the time decomposition cannot compete with a standard Monte Carlo program running on a serial supercomputer. The second problem is memory limitation, which prevents the use of the time decomposition for large systems in which the memory available on a single processor is not large enough to accommodate the entire system. For example, a simulation of an atomic liquid confined in a mesopore with a radius of 200 Å requires over a million atoms; replicating the entire system on each processor of a massively parallel computer would require 12 MB of memory per processor just for the $x$, $y$, and $z$ coordinates of the atoms in the system.

A second parallel algorithm which does not employ a modified Markov chain is the farm algorithm. This approach involves distributing the computational load of a Monte Carlo simulation, calculating the energy of proposed moves, over $P$ available processors, where each processor $P$ is responsible for calculating the contribution to the energy of the proposed move due to some subset of the system's members. Zara and Nicholson[4] have developed a family of five parallel algorithms which embody this approach and applied it to grand canonical Monte Carlo computer simulation (GCMC). In the simplest of these five algorithms, a master processor sends the test coordinates to $P - 1$ server processors (where $P$ is the total number of processors). Each server processor calculates the energy contribution to the test coordinates for its atoms and returns the sum to the master processor, which then determines if the new coordinates are accepted or rejected. The other four algorithms are more complex versions of the first; each employs various methods for saving either the total energies for individual atoms or the energies for individual pair interactions for later reuse. While Zara and Nicholson's work shows that the farm algorithm greatly reduces the total time to determine all pair interactions, their work was limited to a 500-atom system on up to 10 processors and did not include any results for the parallel efficiencies of their algorithms. Jones and Goodfellow[5] have shown that the efficiency

$$E(P) = \frac{C(1)}{PC(P)} \times 100\% \qquad (1)$$

where $E(P)$ and $C(P)$ are the efficiency and elapsed time as a function of the number of processors, $P$, of the farm method applied to $NVT$ Monte Carlo, falls off dramatically with increasing numbers of processors $P$.

Finally, systolic loop methods[6] have been applied to Monte Carlo.[5] However, this approach is efficient only for systems with complex interaction potentials (so that the calculation time greatly exceeds the communication time) and for machines with small numbers of processors and limited memory per processor. Furthermore, it is not easily generalizable to GCMC.

## Parallelizable Grand Canonical Monte Carlo Algorithms

In this work we have developed a spatial decomposition for Monte Carlo that is somewhat similar to Johnson's.[2] This approach, which utilizes a modified Markov chain, is more complicated to program than the Monte Carlo time decomposition discussed earlier but is the only viable approach for MC simulation of very large systems, the subject of planned work. However, the intent of the current work is to demonstrate the new Monte Carlo spatial decomposition, and thus we have also developed a time decomposition program for the purpose of comparison. While we have applied both the spatial and time decompositions to three-dimensional GCMC computer simulation, both are equally applicable to $NVT$ Monte Carlo. GCMC is particularly well suited to study phenomena which occur under conditions in which energy and matter are exchanged between the system and its physical surroundings, such as adsorption, capillary condensation, and vapor–liquid equilibrium. While MD computer simulations have been parallelized[7] and the molecular dynamics method has recently been extended to the grand canonical ensemble (grand canonical molecular dynamics, or GCMD[8–14]), GCMC remains the most efficient method of obtaining equilibrium properties in the grand canonical ensemble. Thus, to model large systems and phenomena which benefit from simulation in the grand canoni-

cal ensemble, such as the preceding example of a liquid confined in a mesopore, a massively parallel GCMC algorithm is needed.

Finally, Shing and Azadipour[15] have developed a new GCMC algorithm that is parallelizable, although it has not been tested on massively parallel hardware. Briefly, this method consists simply of employing a host program to control many *NVT* MC simulations running in parallel, each with different numbers of atoms $N$. While standard GCMC yields equilibrium thermodynamic data for systems at constant chemical potential $\mu$ by varying $N$ and accumulating statistics, Shing and Azadipour's algorithm achieves this by instructing the host program to hop between parallel *NVT* MC simulations with different values of $N$, sampling and accumulating statistics as it goes. Although this novel approach is highly parallelizable, it has the same problems as the time decomposition: Each parallel *NVT* MC simulation must be warmed up and small enough to fit within the memory limitations of the individual processors.

## GCMC Computer Simulation

The GCMC method enables simulation in the grand canonical ensemble (constant chemical potential $\mu$, volume $V$, and temperature $T$) and is discussed more fully in Allen and Tildesley.[16] GCMC employs three types of move: the displacement of a randomly chosen atom, the destruction of a randomly chosen atom, and the creation of a new atom in a randomly chosen location. In a GCMC step, the attempted displacement of a single atom is followed by either an attempted creation or destruction of a single atom, chosen with equal probability.

A proposed displacement of an atom, up to a distance of $\Delta r_{max}$, is accepted if it decreases the system energy:

$$\Delta U < 0 \tag{2}$$

otherwise it is accepted with probability:

$$\exp(-\beta \Delta U) \geq \xi \tag{3}$$

where $\beta = 1/kT$, with $k$ as Boltzmann's constant and $T$ the temperature, $\Delta U$ the resulting change in potential energy for the displacement, and $\xi$ a random number generated uniformly on (0, 1). Throughout this work, $\Delta r_{max} = 0.15\sigma$ was employed, leading to an acceptance percentage of 83% for proposed displacements. An attempted creation at a randomly chosen position in a volume $c$ is accepted if

$$\left(\frac{1}{N(c)+1}\right)\exp\left(\beta\mu(c) + \ln\frac{V(c)}{\Lambda^3} - \beta\Delta U\right) \geq \xi \tag{4}$$

where $V(c)$, $N(c)$, and $\mu(c)$ are the volume of, current number of atoms in, and desired chemical potential of the volume $c$, respectively. In addition, $\Delta U$ is the resulting change in potential energy of creating the atom and $\Lambda$ the de Broglie wavelength. Finally, an attempted destruction in volume $c$ is accepted if

$$N(c)\exp\left(-\beta\mu(c) - \ln\frac{V(c)}{\Lambda^3} - \beta\Delta U\right) \geq \xi \tag{5}$$

The interactions between the atoms in the system were modeled with the short-range cut and shifted Lennard-Jones (LJ) potential:

$$\phi(r) = \begin{cases} \phi^{LJ}(r) - \phi^{LJ}(r^c); & r \leq r^c \\ 0; & r > r^c \end{cases} \tag{6}$$

where $\phi^{LJ}$ is the full LJ potential

$$\phi^{LJ}(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{7}$$

with $r^c$ the cutoff distance, taken to be $2.5\sigma$ throughout this work. A bulk system was simulated by employing periodic boundary conditions applied in the $x$, $y$, and $z$ directions, and two state points were chosen for this work, A and B (see Table I).

## Parallel GCMC: Spatial Decomposition (PGCMC)

The spatial decomposition for grand canonical Monte Carlo simulation takes advantage of the fact

**TABLE I.**
**Decomposition Comparison State Points.**

| State Point | $\frac{\mu'}{kT}$ | $\frac{kT}{\varepsilon}$ | Resulting Density $\rho\sigma^3$ |
|---|---|---|---|
| A | −3.0 | 3.0 | 0.28 |
| B | −2.0 | 1.0 | 0.73 |

$\frac{\mu'}{kT} = \frac{\mu}{kT} + 3\ln\left(\frac{\sigma}{\Lambda}\right)$ [$\varepsilon$ and $\sigma$ from eq. (7)].

that for short-range interaction potentials, processors can work simultaneously in the same volume as long as the regions in which they operate are separated by a distance further than the reach of the potential. This is accomplished by organizing the system volume into domains and assigning one domain per processor. The code is designed so that it can handle noncubic system volumes, which are composed of cubic or noncubic domains, one domain per processor. For example, for a system composed of 64 domains—two, four, and eight domains long in the $x$, $y$, and $z$ directions, respectively—the code employs 64 processors. This is important not only so noncubic system volumes can be studied (the subject of planned work) but also because for much of this comparison study, we employ a hypercube parallel architecture; thus, for numbers of processors which, while powers of 2, are not perfect cubes, noncubic system volumes are required. The corresponding noncubic numbers of domains are assigned to the processors mapped as a three-dimensional mesh embedded in the hypercube.

The domains are subdivided into eight subdomains. An eight-domain subsystem and its corresponding 64 subdomains (eight per domain) is shown in Figure 1. By assigning each processor to work in corresponding subdomains at the same time, and by choosing the subdomain dimension sufficiently large, each processor can perform the three types of GCMC moves—displacement, creation, and destruction—independent of its neighboring processors.

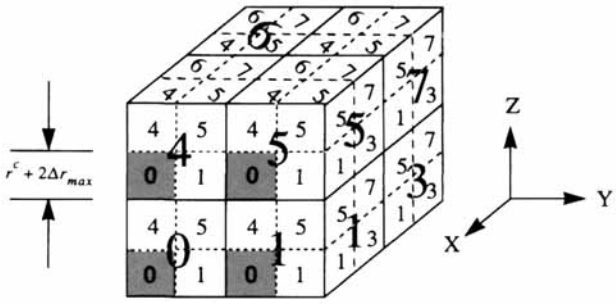


**FIGURE 1.** A subsystem with eight domains and 64 subdomains. Subdomain 0 is marked to show that eight processors responsible for this subvolume are working concurrently in subdomain 0.

## SUBDOMAIN SIZING

The simulation begins with each processor attempting a displacement followed by an attempted creation or destruction in subdomain 0 of the domain for which it is responsible (note the highlighted subdomains 0 in Fig. 1). We can see from Figure 1 that the distance between two subdomains 0 in different domains (and thus on different processors) is the edge length of a single subdomain. Thus, the events occurring in two subdomains 0 in neighboring domains should be independent if the subdomains are sized to be at least $r^c$ (the range of the potential) on edge. However, it is possible that two simultaneous displacements in neighboring domains might occur such that the two atoms move within range of the interaction. For example, consider the example shown in Figure 2. If a proposed displacement of an atom near the edge of subdomain 0 in domain 0 moved it the maximum distance possible, $\Delta r_{max}$, toward subdomain 0 in domain 1, it would land in subdomain 1 in domain 0. Meanwhile, if a simultaneous proposed displacement of an atom near the edge of subdomain 0 in domain 1 placed it $\Delta r_{max}$ deep in subdomain 1 in domain 0, contention between the two proposed moves would occur unless the subdomains were sized greater than $r^c$ + twice the maximum possible displacement on edge. Thus, to remove the possibility of interprocessor contention yet obtain maximum speedup, the subdomains should be sized exactly $r^c + 2\Delta r_{max}$ on edge. Since we have employed values of $\Delta r_{max} = 0.15\sigma$ and $r^c = 2.5\sigma$ throughout this work, the subdomains were sized to be $2.8\sigma$ on edge.

## ENERGY CALCULATION FOR PROPOSED MOVES

The largest computational task for each type of move is the calculation of the energy change for a proposed move. The energy calculation is carried out in parallel as follows. Each processor, working in corresponding subdomains, chooses a set of test coordinates for which the system energy is required. For the case of a creation, the test coordinates represent the proposed atom's location; for a destruction, the coordinates represent an atom proposed for destruction; and for a displacement, the test coordinates will be either an atom's original position or its proposed new position. Each processor first calculates the contribution to its test coordinate's energy due to the atoms residing in the same subdomain as the test coordinate, followed by the energy contribution due to the atoms in the other seven subdomains in the processor's domain. Thus, if all processors are working in subdomain 2 of their domains, the processor

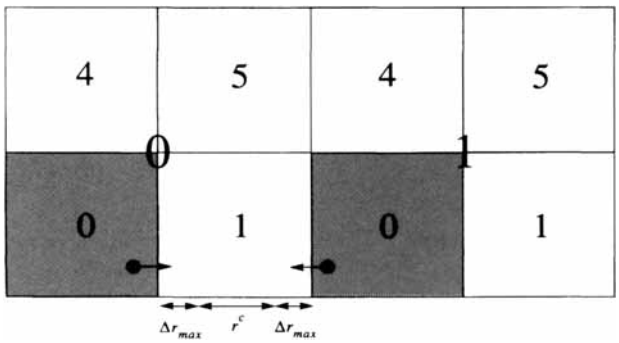


**FIGURE 2.** An example of two atoms in different domains (0 and 1) and thus on different processors, simultaneously moving the maximum possible distance, $\Delta r_{max}$, toward each other yet still remaining beyond the reach of the potential due to the proper sizing of the subdomains: $r^c + 2\Delta r_{max}$ or greater on edge.

responsible for domain 5 in Figure 1 will first calculate the contribution to the subdomain 2 test coordinates due to the atoms in its subdomain 2, followed by those in its subdomains 0, 1, 3, 4, 5, 6, and 7. Each processor then sends its test coordinates to its seven "set I" neighbors—those possessing a subdomain which borders the subdomain containing the test coordinates—and receives the test coordinates from each of its "set II" neighbors—those for which it possesses a subdomain which borders the neighbor's subdomain, which contains the neighbor's test coordinates. After calculating the energy contribution for all of its atoms in the appropriate subdomains and for the test coordinates of all set II neighbors, each processor sends this information to the set II neighbors and receives the corresponding information from its set I neighbors. Each processor is now in possession of the energy of its test coordinates due to the entire system and proceeds to accept or reject the proposed displacement, creation, or destruction.

The seven set I neighboring processors responsible for subdomains which border subdomain $S$ in domain $D$ can be broken down into three groups based on the nature of their border with subdomain $S$ and domain $D$ (see Fig. 3). Three of the processors are responsible for domains which share

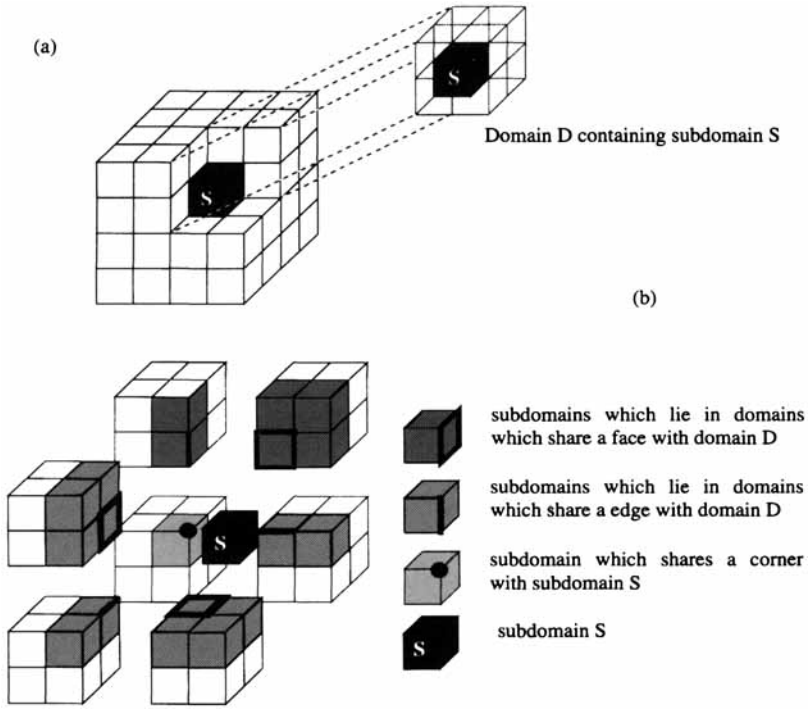


**FIGURE 3.** (a) A subsystem volume with domain $D$ which contains subdomain $S$, and the neighboring seven domains which are contained on subdomain $S$'s seven set I processors. (b) An exploded view of the domains and subdomains of the seven set I processors for subdomain $S$. Subdomain faces which are shared with subdomain $S$ are designated with a bold square, subdomain edges which are shared with subdomain $S$ are designated with a bold line, and the corner which subdomain $S$ shares with a subdomain on one of the seven set I processors is designated with bold sphere.

a face with domain $D$. Each of these three faces accounts for four subdomains, making 12 subdomains that border $S$ and are in domains that share a face with $S$'s domain $D$. Another three processors are responsible for domains which share an edge with domain $D$. Each edge accounts for two subdomains, giving another six subdomains that border $S$ and are in domains that share an edge with $S$'s domain $D$. Finally, one processor is responsible for a subdomain which shares a corner with $S$, bringing the total number of subdomains neighboring $S$ and on the seven set I processors to 19. Counting the seven subdomains which border subdomain $S$ and lie in the same domain $D$ and are thus on the same processor, a grand total of 26 subdomains which border subdomain $S$ is obtained, 19 on seven set I processors and seven on the same processor. The symmetry of this arrangement results in each processor needing the energy contributions of atoms which lie in subdomains on seven other processors for each of its own subdomains (set I processors), as well as each processor providing the energy contributions for the atoms in its domain to seven other processors (set II processors). Two 8-by-7 arrays are used to keep track of these two sets of neighbors for the eight subdomains.

## MONTE CARLO MOVES: CREATIONS, DESTRUCTIONS, AND DISPLACEMENTS

When a processor accepts a proposed creation in one of the subdomains within the domain for which it is responsible, the $x$, $y$, and $z$ coordinates of the atom are simply added to the arrays containing the coordinates of all atoms in that domain. Similarly, for a proposed destruction, the coordinates of the destroyed atom are removed from these arrays. Accepted displacements that occur within a domain are simply handled like an accepted creation (the atom's new position) followed by a destruction (the atom's old position). Since the arrays containing the coordinates are local to each processor, and thus to each domain, displacements that change subdomains while remaining in the same domain can be handled entirely by the processor responsible for a domain. However, occasionally a displacement is accepted for an atom that changes domains and therefore processors. This is handled as follows. When each processor proposes a new set of coordinates for an atom in one of its subdomains (keeping in mind that all processors are working in the correspondingly numbered subdomain within their domain), it checks to see if the new position is outside its domain. Similarly, when each processor calculates the energy contribution of the atoms in its domain to the proposed displacement coordinates of its seven set II neighbors, it checks each set of coordinates to determine if they lie within its domain. After the energy has been calculated for all proposed displacements, each processor determines if its own displacement is accepted or rejected. If a processor determined that its displacement placed the atom outside its domain, it then sends a message to the processor responsible for the destination domain telling whether or not the displacement was accepted. Meanwhile, if a processor determined that the new coordinates of a neighboring processor's displacement landed the atom in its domain, it waits for this message from the neighboring processor regarding whether the displacement was accepted. If the displacement was rejected, the simulation proceeds unchanged; if it was accepted, the processor responsible for the destination domain "takes possession" of the atom simply by adding its coordinates to the arrays containing the coordinates for the atoms in its domain (exactly like a creation), and the processor responsible for the departure domain treats the displacement like a destruction, removing the original coordinates of the atom from its coordinate arrays.

## VALIDATION OF THE PGCMC ALGORITHM

The spatial parallel GCMC program, PGCMC, was validated for several state points by comparing with an $NVT$ molecular dynamics program. This was accomplished by using PGCMC to obtain the number of atoms $N$ for a given set of values for $\mu$, $V$, and $T$. Using $T$ and values of $N$ and $V$ which yielded a corresponding density, $\rho = N/V$, in the $NVT$ molecular dynamics program, the corresponding chemical potential $\mu$ was determined. This value of the state point's chemical potential was then compared to that used as input to PGCMC. The results for an example state point (state point A) are shown in Table II. For this example, the $NVT$ MD simulation employed 2048 atoms, and statistics were accumulated for 80,000 MD timesteps of $\Delta t\sqrt{\varepsilon/m\sigma^2} = 0.01$, after being warmed up from a lattice for 20,000 MD timesteps. The chemical potential was calculated using Widom's method[17] every 10 timesteps. More discussion about the MD technique can be found in Allen and Tildesley.[16]

**TABLE II.**
**An Example Comparison between PGCMC and *NVT* MD for State Point A.**

| State Variable | PGCMC (* denotes an independent variable) | *NVT* MD (* denotes an independent variable) |
|---|---|---|
| $\frac{\mu'}{kT}$ | −3.0* | −2.97 |
| $\rho\sigma^3 = \frac{N}{V}$ | 0.278 | 0.278* |
| $N$ | 50,050.78 | 2048* |
| $V$ | 179,830.784* | 7359.41* |
| $\frac{kT}{\varepsilon}$ | 3.0* | 3.0* |

## Parallel GCMC: Time Decomposition (tPGCMC)

As stated in the Introduction, the GCMC time decomposition simply involves directing each processor of the parallel computer to perform an identical and simultaneous GCMC simulation while starting with an independent random number generator seed. Thus each processor performs the predetermined number of steps—an attempted displacement followed by either a creation or destruction attempt—for its simulation volume. After each step, the thermodynamic properties, such as the potential energy and number of atoms for the system volume, are averaged across all processors and stored by processor 0 for postrun evaluation. Linked cells[16] were employed to minimize the runtime.

## Efficiency and Comparison Methods

While it is a simple matter to determine the computational speed of both the spatial (PGCMC) and time decomposition (tPGCMC) algorithms, it is much more difficult to determine the overall efficiency of the algorithms, which is a combination of computational speed and statistical efficiency. We have accomplished this by measuring two parameters for each algorithm: the elapsed time per GCMC step and the number of GCMC steps required for the data to reach a given value of the standard error. By multiplying these two parameters, we obtain a measurement of the time necessary for each algorithm to reach a given value of the standard error (i.e., the time to obtain data with a given statistical quality).

For the purposes of comparison between the spatial and time decompositions, we ran exactly the same system sizes for each algorithm. The system sizes are contained in Tables III and IV, in which we present the scaling results for the spatial and time decompositions, respectively. The results for the time decomposition are limited to runs on

**TABLE III.**
**Scaling Results for the Spatial Decomposition (PGCMC).**

| Number of Processors $P$ | System Dimensions $\frac{x}{\sigma}$ | $\frac{y}{\sigma}$ | $\frac{z}{\sigma}$ | System Volume $\frac{V}{\sigma^3}$ | State Point A Ave. No. of Atoms $\langle\langle N\rangle\rangle$ | State Point A Elapsed Time s / step | State Point B Ave. No. of Atoms $\langle\langle N\rangle\rangle$ | State Point B Elapsed Time s / step |
|---|---|---|---|---|---|---|---|---|
| 8 | 11.2 | 11.2 | 11.2 | 1404.928 | 392 | 0.041 | | |
| 16 | 22.4 | 11.2 | 11.2 | 2809.856 | 783 | 0.042 | | |
| 32 | 22.4 | 22.4 | 11.2 | 5619.712 | 1,566 | 0.043 | 4126 | 0.071 |
| 64 | 22.4 | 22.4 | 22.4 | 11239.424 | 3,131 | 0.043 | 8231 | 0.072 |
| 128 | 44.8 | 22.4 | 22.4 | 22478.848 | 6,254 | 0.044 | 16492 | 0.072 |
| 256 | 44.8 | 44.8 | 22.4 | 44957.696 | 12,519 | 0.044 | 32910 | 0.072 |
| 512 | 44.8 | 44.8 | 44.8 | 89915.392 | 25,043 | 0.045 | | |
| 1024 | 89.6 | 44.8 | 44.8 | 179830.784 | 50,047 | 0.045 | | |

**TABLE IV.**
**Scaling Results for the Time Decomposition (tPGCMC).**

| Number of Processors $P$ | System Volume $\frac{V}{\sigma^3}$ | State Point A | | State Point B | |
|---|---|---|---|---|---|
| | | Ave. No. of Atoms $\langle\langle N \rangle\rangle$ | Elapsed Time s / step | Ave. No. of Atoms $\langle\langle N \rangle\rangle$ | Elapsed Time s / step |
| 8 | 1404.928 | 391 | 0.020 | | |
| 16 | 2809.856 | 776 | 0.025 | | |
| 32 | 5619.712 | 1,559 | 0.030 | 4114 | 0.048 |
| 64 | 11239.424 | 3,118 | 0.050 | 8179 | 0.074 |
| 128 | 22478.848 | 6,231 | 0.081 | 16411 | 0.097 |

128 and fewer processors; larger systems were impractical to run given warm-up time limitations.

The data used in the statistical analysis were generated as follows. Both programs were modified to output the simulation potential energy $E$ after each Monte Carlo move. For the spatial decomposition, this modification consists simply of having each processor determine the potential energy of the atoms in its domain at the start of the program. This information is then summed over all processors, yielding the total system potential energy, and saved on processor 0. After each attempted Monte Carlo move, the change in potential energy due each processor's move (zero if the move was rejected) is summed across all processors, added to the previous value of the total system's potential energy $E$ on processor 0, and saved in an output file. Thus, for the spatial decomposition, the simulation potential energy data points $E$ represent the evolution of the total system's potential energy.

For the time decomposition, after each attempted Monte Carlo move, the potential energy of each processor's system was summed across all processors, divided by the number of processors (yielding an average potential energy for the parallel simulation), and saved in an output file. Therefore, the simulation potential energy $E$ for the time decomposition represents the average system potential energy for the $P$ parallel simulations occurring concurrently on $P$ processors.

The simulation potential energy data files can then be statistically analyzed as follows. The raw data points for the simulation potential energy $E$ were first "binned" to filter the effects of the correlation length. That is, the raw data, numbering $N_{\text{pts}}$, were divided into $B$ bins, with each bin containing $b = N_{\text{pts}}/B$ raw data points. The contents of each bin were then averaged, yielding a set of $B$ refined data points $F$. These refined data were then used to calculate the standard error (SE) as a function of the number of refined data points used $f$:

$$\text{SE}(f) = \frac{\sum_{i=1}^{f}(F_i - F_{\text{ave}})^2}{f(f-1)} \tag{8}$$

where

$$F_{\text{ave}} = \frac{\sum_{i=1}^{f} F_i}{f} \tag{9}$$

The standard error can then be plotted as a function of the number of refined data points $f$ used for its calculation, as in the example shown in Figure 4. For this example, the data were generated by the spatial decomposition program PGCMC running on 1024 processors for state point A. The data shown were produced by refining 960,000 raw data points ($N_{\text{pts}} = 960{,}000$) by binning them into 120 bins ($B = 120$), each bin having 8000 data points ($b = 8000$). The 8000 points in each bin were then averaged, yielding 120 refined data points ($F = 120$). The standard error was calculated as a function of the number ($f$) of the included refined data points and plotted against $f$. It is easily seen from this figure that with increasing $f$, the number of refined data points employed in determination of the standard error, the standard error monotonically decreases, a direct result of the fact that increasing the length of the simulation increases the quality of the statistics of the simulation potential energy.

By determining the number of refined data points required to reach a given value of the standard error, $f_{SE}$, the total number of raw data points required to reach this point can be obtained by

$$N_{pts}^{SE} = (f_{SE})(b) \tag{10}$$

For sufficiently large bin sizes, the effects of the correlation length will be removed; thus the number of raw data points required to reach a given value of the standard error, $N_{pts}^{SE}$, will be independent of bin size $b$. While past experience has shown that for some Monte Carlo simulations, $b$ must often be chosen five to six times the length of the correlation,[18] for this work we simply employed a trial-and-error method[18] of choosing increasingly large values of $b$ until reaching the range at which $N_{pts}^{SE}$ is independent of $b$.

Using calls to timing routines, the time per GCMC step, $C_{step}$ (a GCMC step is defined as each processor attempting a displacement and then either a creation or destruction), can be determined for each decomposition. Since each GCMC step generates two raw data points, the time to achieve a given value of the standard error, $C_{SE}$, can be determined from

$$C_{SE} = \frac{N_{pts}^{SE} C_{step}}{2} \tag{11}$$

## Performance and Optimization

While both the spatial (PGCMC) and time (tPGCMC) decompositions have also been implemented on Sandia's 1872 processor Intel Paragon, most of this work was carried out with the versions of both codes implemented and optimized for Sandia's 1024 processor nCUBE2 massively parallel computer. The nCUBE2 is a hypercube topology, distributed memory, multiple instruction stream, multiple data stream (MIMD) computer. The individual nCUBE processor is a single chip very large scale integration (VLSI) implementation of nCUBE's proprietary instruction set architecture, integrating both communications and memory control. The 28 direct memory access (DMA) channels used to implement the hypercube interconnect can operate independently in parallel with each other and in parallel with computation, limited ultimately only by their contention for shared memory bandwidth. The channels implement "wormhole" routing, which reduces the performance impact of communicating to non-nearest-neighbor processors.

Programs for the nCUBE2 are implemented in FORTRAN, C, or C++. All coordination or cooperation between processors (i.e., parallel processing) is performed via explicit message passing calls. Message writes are nonblocking because the

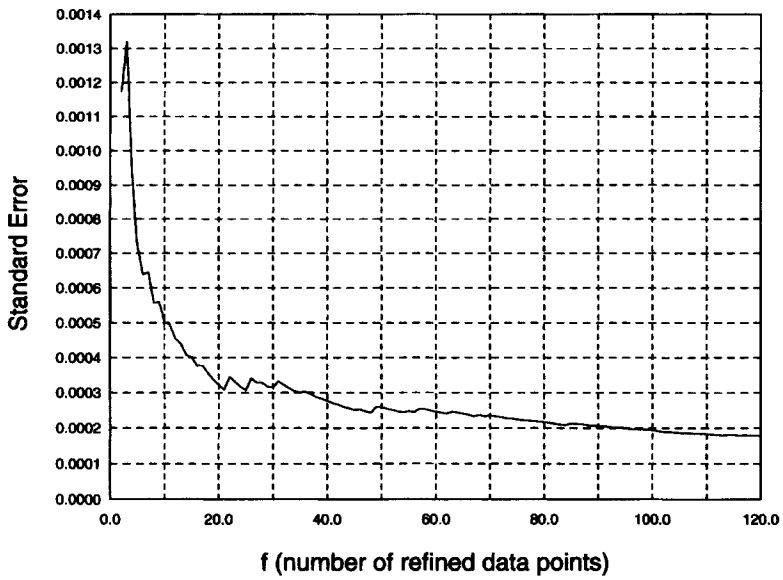


**FIGURE 4.** A standard error plot for data generated by the spatial decomposition program PGCMC running on 1024 processors (state point A, bin size = 8000). The standard error (SE) monotonically decreases with increasingly long simulations (increasing $f$), as expected.

data are copied to a communication buffer and control is returned to the program. Message reads are blocking.

PGCMC is implemented in FORTRAN. Performance is optimized by attempting to overlap communications both with other communications and with computation. The parallel transfer capabilities of the DMA channels are exploited by issuing several of the nonblocking writes in a row and are handled by the operating system transparently to the user. By issuing the message writes as soon as possible and delaying the message reads with computation as long as possible, communication may sometimes be completely hidden behind computation. This technique is exploited in the energy calculations by first writing the test coordinates to all seven of the set I neighbors. Then the energy contributions of the processor's own subdomains are calculated. The requests from the seven set II processors are then read and worked on, and results are written back one at a time. Finally, the energy contributions of the other processors are read. This effectively uses the DMA channels in parallel and hides the message writing of the test coordinates behind computation. We are currently testing a refinement of dividing the calculation of the contribution of the processor's own subdomains to the processor's own test coordinates into two parts, and using the second part to attempt to hide the communications of the results.

PGCMC uses nCUBE supplied routines for embedding a three-dimensional mesh in the hypercube in such a way that all communication between neighboring domains along the axis of the mesh and across periodic boundaries are on processors that are also directly connected nearest neighbors in the hypercube. This results in all communications to processors whose subdomains share a face being "1-hop" or nearest neighbor. All communications to processors whose subdomains share just edges are 2-hop, and those to processors sharing just a corner are 3-hop. This minimizes the potential contention for channels.

## Results and Discussion

To investigate the scaling characteristics of both algorithms, timings of varying system sizes and number of processors for the state points in Table I were obtained. Each system, regardless of size, was warmed up and the density was checked to be sure the system was fully equilibrated. The number of warm-up steps necessary was not only a function of density but also of algorithm. The total central processing unit (CPU) time for warm-up for both algorithms and state points was estimated by multiplying the number of warm-up steps required by the algorithm's measured speed during the production phase (Table V). However, these warm-up times are estimates because our GCMC simulations began from a single atom and were warmed up to the equilibrium number of atoms. The system sizes and scaling results for the spatial and time decompositions for both state points, shown in Tables III and IV, are plotted in Figure 5.

**TABLE V.**
**Warm-Up Data.**

| | State Point A | | | | State Point B | | | |
|---|---|---|---|---|---|---|---|---|
| | PGCMC | | tPGCMC | | PGCMC | | tPGCMC | |
| Number of Processors $P$ | Number of Warm-up Steps | Approx. Warm-up CPU Time (hours) | Number of Warm-up Steps | Approx. Warm-up CPU Time (hours) | Number of Warm-up Steps | Approx. Warm-up CPU Time (hours) | Number of Warm-up Steps | Approx. Warm-Up CPU Time (hours) |
| 8 | 80,000 | 0.91 | 80,000 | 0.44 | | | | |
| 16 | 80,000 | 0.93 | 80,000 | 0.56 | | | | |
| 32 | 80,000 | 0.96 | 80,000 | 0.67 | 80,000 | 1.6 | 1,600,000 | 21.2 |
| 64 | 80,000 | 0.96 | 80,000 | 1.1 | 80,000 | 1.6 | 3,920,000 | 81.0 |
| 128 | 80,000 | 0.98 | 80,000 | 1.8 | 160,000 | 3.2 | 7,280,000 | 196.2 |
| 256 | 80,000 | 0.98 | | | 160,000 | 3.2 | | |
| 512 | 80,000 | 1.0 | | | | | | |
| 1024 | 80,000 | 1.0 | | | | | | |

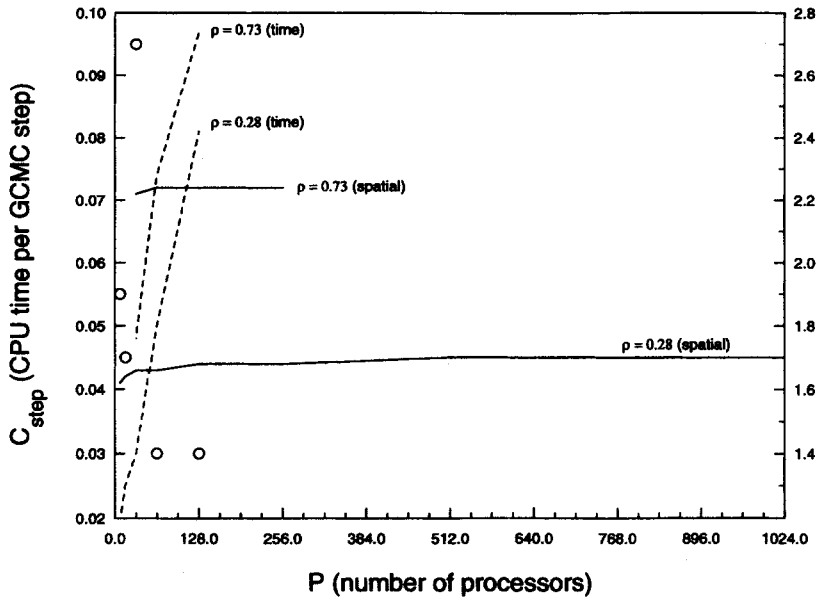


**FIGURE 5.** Scaling results for the spatial decomposition tPGCMC (solid line) and time decomposition tPGCMC (dashed line) plotted against the left-hand axis. While PGCMC scales with the number of processors (and hence system volume), the increase in $C_{step}$ with increasing numbers of processors $P$ for tPGCMC is due to the fact that this decomposition does not scale with system size. The circles, plotted against the right-hand axis, represent the decomposition comparison ratio $R$ for state point A plotted against the number of processors $P$ (and hence system size).

From the results plotted in Figure 5, we can see that the spatial GCMC algorithm scales nearly exactly with system size for both state points. However, we can see from Table III that the time per step ($C_{step}$) of the spatial decomposition does increase slightly with the number of processors. This is due simply to the fact that the interprocessor communication necessary to gather the thermodynamic data on processor 0 for print-out to the data file (done with logarithmic "fold" routines)[19]—a very small part of the overall simulation time—increases with increasing numbers of processors. From Figure 5 we also see that the time decomposition requires increasingly more time per step ($C_{step}$) as system size increases, regardless of the state point density. While this increase in $C_{step}$ for the time decomposition is partly due to increased communication costs for larger numbers of processors (to average the potential energy across $P$ simulations on $P$ processors), the increase is primarily due to the increase in system size with increasing $P$. The increase in $C_{step}$ may well be worthwhile: A measure of $C_{step}$ provides only half of the information necessary to answer the question of which decomposition is more efficient; we still need the relative number of GCMC steps to obtain a given value of the standard error.

As discussed earlier, once the time per step, $C_{step}$, is known, it can be multiplied by a measure of the number of steps to reach a given value of the standard error to yield the amount of elapsed time to reach that value of the standard error, $C_{SE}$. For the purposes of demonstrating how the overall efficiency of the spatial decomposition scales with system size (and therefore numbers of processors), we have plotted the speedup $S$

$$S(P) = \frac{C_{SE}(P = 8)}{C_{SE}(P)} \tag{12}$$

and the overall efficiency $E$

$$E(P) = \frac{S}{P/8} \times 100\% \tag{13}$$

as a function of the number of processors (divided by 8) in Figure 6 for PGCMC at state point A. The data for these plots are contained in Table VI. Note that the definitions of both the speedup and the

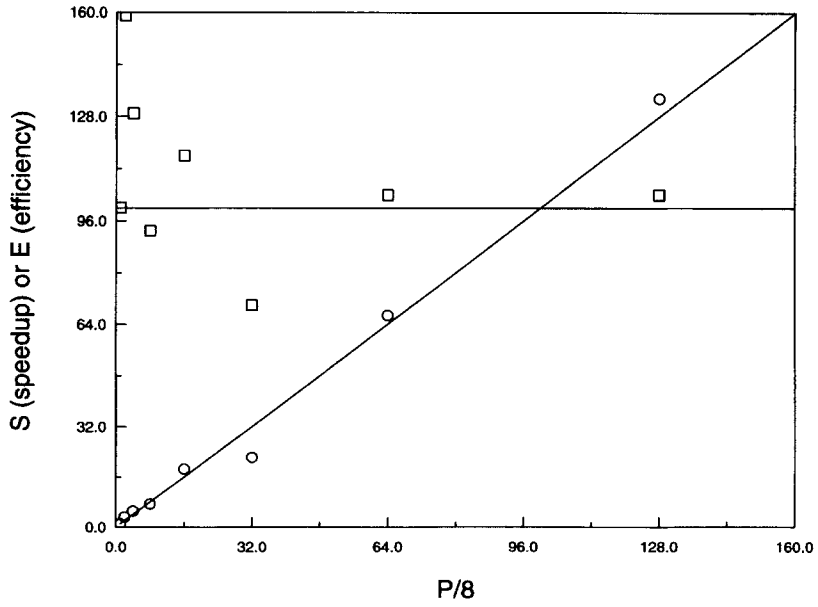


**FIGURE 6.** Speedup (circles) and efficiency [squares, as per eq. (13)] for the spatial decomposition plotted against the number of processors divided by 8 (the smallest number of processors the PGCMC program runs on is 8). The fluctuations in efficiency are due to the statistical nature of the evaluation of the efficiency. The ideal speedup of $S = P/8$ is represented by the sloped line, and 100% efficiency is shown by the flat line.

efficiency are based on a comparison between results for $P$ processors and eight processors, as opposed to a single processor ($P = 1$); hence the use of $P/8$ for the abscissa. This is because PGCMC runs on a minimum of eight processors. Furthermore, we can see from this figure that for some system sizes (numbers of processors), the PGCMC algorithm actually experiences superlinear speedup and thus efficiencies greater than 100%. These results are due to scatter in the data and thus the

**TABLE VI.**
**Speedup and Efficiency Results for Spatial Decomposition (PGCMC) for State Point A.**

| Number of Processors $P$ | Number of GCMC Steps | Number of Raw Data Points $N_{pts}$ | Bin Size Used to Refine Raw Data $b$ | Number of Raw Data Points to a Standard Error of 0.0008 $N_{pts}^{SE=0.0008}$ | Elapsed Time to a Standard Error of 0.0008 (hours) | Speedup [eq. (15)] $S$ | Efficiency [eq. (16)] $E$ |
|---|---|---|---|---|---|---|---|
| 8 | 3200K | 6400K | 12500 | 5862.5K | 33.4 | 1.0 | 100% |
| 16 | 2400K | 4800K | 12000 | 1792K | 10.5 | 3.2 | 159% |
| 32 | 2400K | 4800K | 12000 | 1088K | 6.5 | 5.1 | 129% |
| 64 | 2400K | 4800K | 8000 | 760K | 4.5 | 7.4 | 93% |
| 128 | 1920K | 3840K | 8000 | 288K | 1.8 | 18.6 | 116% |
| 256 | 960K | 1920K | 8000 | 248K | 1.5 | 22.3 | 70% |
| 512 | 400K | 800K | 8000 | 80K | 0.50 | 66.8 | 104% |
| 1024 | 480K | 960K | 8000 | 40K | 0.25 | 133.6 | 104% |

**TABLE VII.**
**Decomposition Comparison Data for State Point A.**

| Number of Processors $P$ | Decomposition: Spatial (PGCMC) or Time (tPGCMC) | Number of GCMC Steps | Number of Raw Data Points $N_{pts}$ | Bin Size Used to Refine Raw Data $b$ | Standard Error Used in PGCMC / tPGCMC Comparison SE | Number of Raw Data Points to Standard Error $N_{pts}^{SE}$ | Decomposition Comparison Ratio $R$ |
|---|---|---|---|---|---|---|---|
| 8 | Spatial | 3200K | 6400K | 12500 | 0.0009 | 4512.5K | 1.9 |
| | Time | 2400K | 4800K | 32000 | | 4736K | |
| 16 | Spatial | 2400K | 4800K | 12000 | 0.0006 | 4512K | 1.7 |
| | Time | 2400K | 4800K | 32000 | | 4512K | |
| 32 | Spatial | 2400K | 4800K | 12000 | 0.0005 | 4164K | 2.7 |
| | Time | 2000K | 4000K | 32000 | | 2176K | |
| 64 | Spatial | 2400K | 4800K | 8000 | 0.0004 | 2944K | 1.4 |
| | Time | 2400K | 4800K | 32000 | | 1760K | |
| 128 | Spatial | 1920K | 3840K | 8000 | 0.0003 | 2432K | 1.4 |
| | Time | 760K | 1520K | 20000 | | 920K | |

statistical nature of the evaluation of the speedup and efficiency.

By calculating a decomposition comparison ratio $R$

$$R = \frac{C_{SE}^{spatial}}{C_{SE}^{time}} \tag{14}$$

for a given value of standard error, we can make a comparison between the relative overall efficiencies of the spatial and time decompositions. Because of the large spread in the data, values of $R$ have been calculated for differing standard errors for each system size (number of processors). The number of GCMC sweeps, raw data points, bin sizes used to refine the raw data for both decompositions at state point A, as well as the standard error used to calculate $R$, the number of raw data points required to reach this value of the standard error for both decompositions, and $R$ itself are tabulated in Table VII. The decomposition comparison ratio $R$ is plotted against the number of processors using the right-hand axis in Figure 5. While the data are widely scattered, the overall efficiency of the time decomposition is better than that of the spatial decomposition for this state point (A).

State point A represents a supercritical fluid with a density, ( $\rho\sigma^3 = 0.28$), which is at neither end of the spectrum of possible simulation densities. At higher densities the relative number of atoms per processor would seem to favor the spatial decomposition; thus we have tested each decomposition at a higher density, $\rho\sigma^3 = 0.73$ (state point B). However, rather than calculate a decomposition comparison ratio, we have simply plotted standard error as a function of required CPU time for varying system sizes (and thus numbers of processors ) in Figure 7. The bin sizes used to refine the raw data were 50,000 for all results in this figure. From this figure, which again reflects not only the speed of each algorithm but also its efficiency at sampling phase space, we can see that even at this higher density, the time decomposition is superior to the spatial decomposition. In fact, the time decomposition on 32 processors out performs the spatial decomposition not only on 32 processors, but also on 256 processors. However, at this density the warm-up time necessary for the time decomposition rapidly increases with increasing system size (Table V). If we include the estimated warm-up time, these results change dramatically (Fig. 7 inset) .

## Conclusions and Future Work

We have shown that the spatial grand canonical Monte Carlo algorithm scales with system size and is thus well suited for studying large open systems of atoms with short-range interactions (e.g., fluids in mesopores). While the overall efficiency of the time decomposition is superior to that of the spatial decomposition, the spatial decomposition is

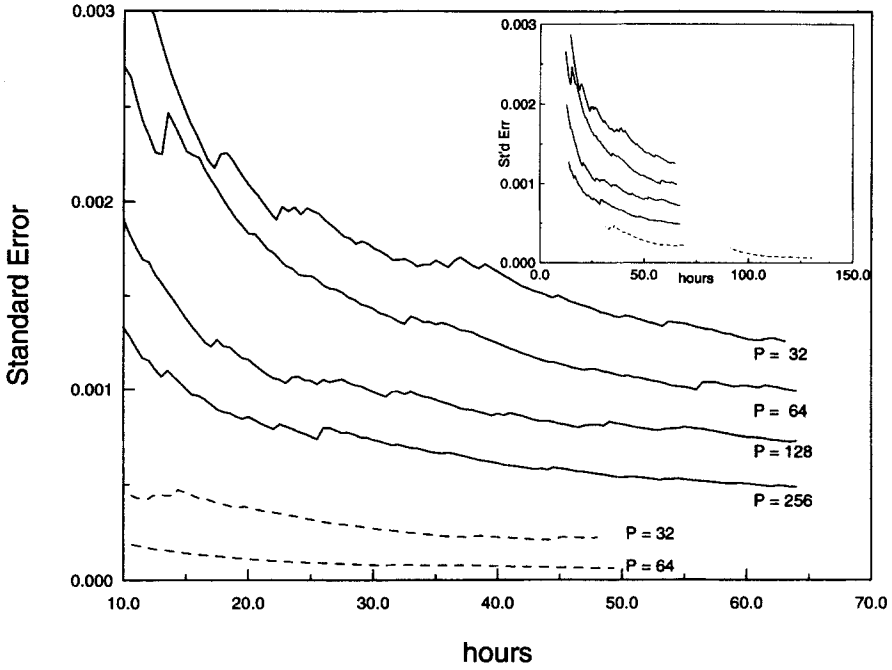


**FIGURE 7.** The standard error as a function of CPU time for the spatial decomposition (solid lines) and the time decomposition (dashed lines) for state point B. The standard error is also plotted against the total CPU time (warm-up + production) in the inset.

preferred for larger systems due to the warm-up time requirements of the time decomposition. Perhaps the best approach for achieving an efficient parallel Monte Carlo simulation might be to employ the spatial decomposition to achieve an equilibrated system and switch to the time decomposition for the production phase. While this approach would limit system size to that which would fit within the memory available on individual processors of parallel computers, this limitation is rapidly becoming less restrictive as the amount of memory per processor increases.

Several outstanding questions about the spatial GCMC algorithm remain. For example, would a more efficient simulation be obtained if, instead of sequencing each processor through subdomains in order 0 through 7, the sequence was selected randomly each GCMC step with a global random number sequence? The cost of this alternative is minimal since no extra communication is required beyond a one-time broadcast of the initial global random number seed. Furthermore, the spatial algorithm might be significantly speeded up by implementing further performance optimizations, such as neighbor lists and acceptance criteria using $N$ and $V$ for cells larger than subdomains, and entirely avoiding energy calculations in some subdomains by determining if the test position is out of range of the whole subdomain. Finally, we intend to enable the algorithm to handle larger volumes efficiently on a given set of processors by implementing more than one domain per processor.

## Acknowledgments

The authors thank John DeLaurentis and Richard Fye, Sandia National Laboratories, and Brian Peterson, Mobil Research and Development Corporation, for several helpful discussions. This work was performed at Sandia National Laboratories, which is operated for the U.S. Department of Energy (DOE) under contract number DE-AC04-94AL85000.

## References

1. G. Heffelfinger, *Proceedings of the 1992 DAGS/PC Symposium*, Dartmouth Institute for Advanced Graduate Studies, Hanover, 229 (1993).
2. M. A. Johnson, *Concurrent Computation and Its Application to the Study of Melting in Two Dimensions*, Thesis, California Institute of Technology, 1986.
3. D. J. Adams, *J. Comp. Phys.*, **75**, 138 (1988).
4. S. J. Zara and D. Nicholson, *Molec. Simul.*, **5**, 245 (1990).

5. D. M. Jones and J. M. Goodfellow, *J. Comp. Chem.*, **14,** 127 (1993).
6. R. A. Whiteside, P. G. Hibbard, and N. S. Ostland, *Proceedings of the Third International Conference on Distributed Systems,* IEEE Computer Society, New York, 800 (1982).
7. S. J. Plimpton, *J. Comp. Phys.*, **117,** 1 (1995).
8. G. S. Heffelfinger and F. van Swol, *J. Chem. Phys.*, **100,** 7548 (1994).
9. T. Cagin and B. M. Pettitt, *Molec. Simul.*, **6,** 5 (1991).
10. T. Cagin and B. M. Pettitt, *Molec. Phys.* **72,** 169 (1991).
11. J. Ji, T. Cagin, and B. M. Pettitt, *J. Chem. Phys.*, **96,** 1333 (1992).
12. J. E. Mertz and B. M. Pettitt, *Supercomp. Appl. High Perform. Comp.*, **8,** 47 (1994).
13. C. Lo and B. Palmer, *J. Chem. Phys.*, **102,** 925 (1995).
14. L. F. Vega, K. S. Shing, and L. F. Rull, *Molec. Phys.*, **82,** 439 (1994).
15. K. S. Shing and S. R. Azadipour, *Chem. Phys. Lett.*, **190,** 386 (1992).
16. M. P. Allen and D. J. Tildesley, *Computer Simulations of Liquids,* Clarendon Press, Oxford, 1987.
17. B. Widom, *J. Chem. Phys.*, **39,** 2808 (1963).
18. R. M. Fye, Sandia National Laboratories, private communication.
19. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors,* Vol. 1, Prentice Hall, Englewood Cliffs, NJ, 1988.